

Introduction to Embedded System Design
Professor. Dhananjay V. Gadre
Netaji Subash University of Technology
Professor. Badri Subudhi
Indian Institute of Technology, Jammu
Lecture 13

MSP430 Architecture- Continued And Introduction to Lunchbox

Hello and welcome back to a new session. In this session we are going to talk about the MSP430 microcontroller, continuing from our previous lecture. This is part of the online course on Introduction to Embedded System Design and as usual I am your instructor Dhananjay Gadre at Netaji Subash University of Technology. Now in the last lecture on MSP430, we looked at the MSP430 series map of the memory map and specifically the G2553 microcontroller we were looking at that here.

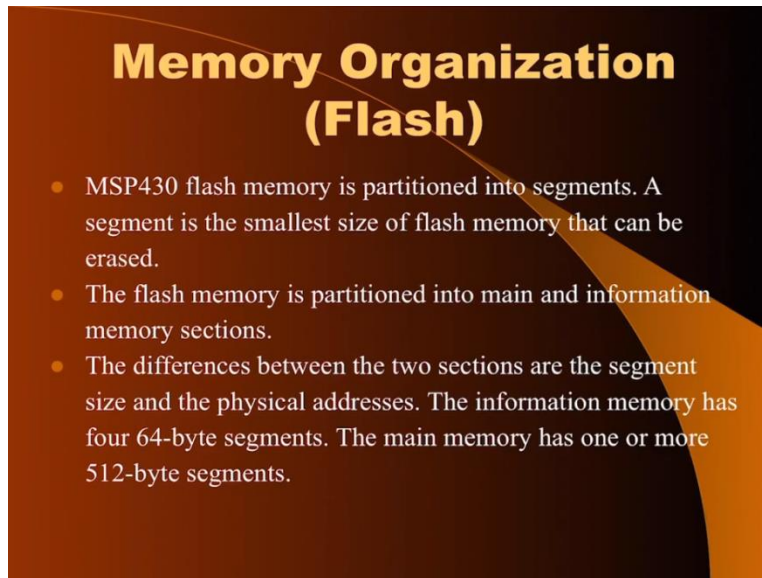
(Refer Slide Time: 01:02)

MSP430 G Series Memory Map

		MSP430G2153	MSP430G2253 MSP430G2213	MSP430G2353 MSP430G2313	MSP430G2453 MSP430G2413	MSP430G2553 MSP430G2513
Memory	Size	1kB	2kB	4kB	8kB	16kB
Main: interrupt vector	Flash	0xFFFF to 0xFFC0	0xFFFF to 0xFFC0	0xFFFF to 0xFFC0	0xFFFF to 0xFFC0	0xFFFF to 0xFFC0
Main: code memory	Flash	0xFFFF to 0xFC00	0xFFFF to 0xF800	0xFFFF to 0xF000	0xFFFF to 0xE000	0xFFFF to 0xC000
Information memory	Size	256 Byte	256 Byte	256 Byte	256 Byte	256 Byte
	Flash	010FFh to 01000h	010FFh to 01000h	010FFh to 01000h	010FFh to 01000h	010FFh to 01000h
RAM	Size	256 Byte	256 Byte	256 Byte	512 Byte	512 Byte
		0x02FF to 0x0200	0x02FF to 0x0200	0x02FF to 0x0200	0x03FF to 0x0200	0x03FF to 0x0200
Peripherals	16-bit	01FFh to 0100h	01FFh to 0100h	01FFh to 0100h	01FFh to 0100h	01FFh to 0100h
	8-bit	0FFh to 010h	0FFh to 010h	0FFh to 010h	0FFh to 010h	0FFh to 010h
	8-bit SFR	0Fh to 00h	0Fh to 00h	0Fh to 00h	0Fh to 00h	0Fh to 00h

Let us continue further.

(Refer Slide Time: 01:06)



Memory Organization (Flash)

- MSP430 flash memory is partitioned into segments. A segment is the smallest size of flash memory that can be erased.
- The flash memory is partitioned into main and information memory sections.
- The differences between the two sections are the segment size and the physical addresses. The information memory has four 64-byte segments. The main memory has one or more 512-byte segments.

The flash memory which in this case is 16 Kilobytes of flash memory is partitioned into segments. A segment is the smallest amount of memory that this microcontroller can erase and the flash memory is partitioned in two parts, as we discussed, one is the memory where you store the code and the other is what is called as the information memory.

The differences between the two sections here, the code segment and the information segment is the size of the segment. The information segment is 64 bytes segments and the main memory where you store code is 512 bytes.

(Refer Slide Time: 01:51)

Memory Organization (Flash) FFFE; FFFF

- The start address of Flash/ROM depends on the amount of Flash/ROM present and varies by device.
- The end address for Flash/ROM is 0xFFFF for devices with less than 60KB Flash/ROM.
- Flash can be used for both code and data but is generally used as code memory. The code memory holds the program.
- The interrupt vectors are used to handle the interrupts. The interrupt vector table is mapped into the upper 16 words of Flash/ROM address space, with the highest priority interrupt vector at the highest Flash/ROM word address (0xFFFE).

0xFFFF	Interrupt and Reset Vector Table
0xFFCD	
0xFFBF	Flash Code Memory (lower boundary varies)
0xF000	
0xEFFF	Flash Information Memory
0x1100	
0x10FF	Bootstrap Loader (BSL)
0x1000	
0x0FFF	RAM (upper boundary varies)
0x0C00	
0x0BFF	peripheral registers (word access)
0x0280	
0x027F	peripheral registers (byte access)
0x0200	
0x01FF	special function registers (byte access)
0x0100	
0x00FF	
0x0100	
0x000F	
0x0000	

In our case the amount of memory that is available you have, in 2553 we have 16 kilobytes of memory. We have already discussed what all we can do with this and I will repeat here that the flash memory can be used to store code as well as constants and you can modify some of these constants also. The interrupt vectors are stored in the flash memory part of the code code memory in the very upper the is the uppermost part of the flash memory.

This is the where you are storing the interrupts and various interrupt vectors as well as the reset vectors. We have seen that the reset vectors is at address let me write it down here, is at address FFFE and FFFF. You see it takes two bytes because the addresses are 16 bits that is why you have to store these two locations.

(Refer Slide Time: 03:08)

8.4 Interrupt Vector Addresses

The interrupt vectors and the power-up starting address are located in the address range 0FFFh to 0FFC0h. The vector contains the 16-bit address of the appropriate interrupt handler instruction sequence. If the reset vector (located at address 0FFFh) contains 0FFFh (for example, flash is not programmed), the CPU goes into LPM4 immediately after power-up.

Table 5. Interrupt Sources, Flags, and Vectors

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-Up External Reset Watchdog Timer+ Flash key violation PC out-of-range ⁽¹⁾	PORIFG RSTIFG WDTIFG KEYV ⁽²⁾	Reset	0FFFh	31, highest
NMI Oscillator fault Flash memory access violation	NMIIFG OFIFG ACCVIFG ⁽²⁾⁽³⁾	(non)-maskable (non)-maskable (non)-maskable	0FFCCh	30
Timer1_A3	TA1CCR0 CCFIFG ⁽⁴⁾	maskable	0FFFAh	29
Timer1_A3	TA1CCR2 TA1CCR1 CCIFG, TAIFG ⁽⁴⁾⁽⁵⁾	maskable	0FFF8h	28
Comparator_A+	CAIFG ⁽⁶⁾	maskable	0FFF6h	27
Watchdog Timer+	WDTIFG	maskable	0FFF4h	26
Timer0_A3	TA0CCR0 CCFIFG ⁽⁴⁾	maskable	0FFF2h	25
Timer0_A3	TA0CCR2 TA0CCR1 CCIFG, TAIFG (6)(4)	maskable	0FFF0h	24
USCI_A0/USCI_B0 receive USCI_B0 PC status	UCA0RXIFG, UCB0RXIFG ⁽⁷⁾⁽⁸⁾	maskable	0FFECh	23
USCI_A0/USCI_B0 transmit USCI_B0 PC receive/transmit	UCA0TXIFG, UCB0TXIFG ⁽⁷⁾⁽⁸⁾	maskable	0FFECh	22
ADC10 (MSP430G2x53 only)	ADC10IFG ⁽⁴⁾	maskable	0FFEAh	21
			0FFE8h	20
I/O Port P2 (up to eight flags)	P2IFG.0 to P2IFG.7 ⁽²⁾⁽⁴⁾	maskable	0FFE6h	19
I/O Port P1 (up to eight flags)	P1IFG.0 to P1IFG.7 ⁽²⁾⁽⁴⁾	maskable	0FFE4h	18
			0FFE2h	17
			0FFE0h	16
See ⁽⁷⁾			0FFDEh	15
See ⁽⁸⁾			0FFDEh to 0FFC0h	14 to 0, lowest

(1) A reset is generated if the CPU tries to fetch instructions from within the module register memory address range (0h to 01FFh) or from within unused address ranges.
 (2) Multiple source flags.
 (3) (non)-maskable: the individual interrupt-enable bit can disable an interrupt event, but the general interrupt enable cannot.
 (4) Interrupt flags are located in the module.
 (5) In SPI mode: UCB0RXIFG, in PC mode: UCA0RXIFG, UCA0TXIFG, UCSTPIFG.
 (6) In UART or SPI mode: UCB0TXIFG, in PC mode: UCA0RXIFG, UCA0TXIFG.
 (7) This location is used as bootstrap loader security key (BSL_SKEY). A 0xA555 at this location disables the BSL completely. A zero (0h) disables the execution of the flash if an invalid password is supplied.
 (8) The interrupt vectors at addresses 0FFDEh to 0FFC0h are not used in this device and can be used for regular program code if necessary.

Here is the table which describes all the interrupt vectors. G2553 supports 32 interrupts and each of the interrupt requires interrupt sub routine. Interrupt sub routine is going to be a program in the flash memory and in this vector table you store the address of that location in the flash memory and so each address is 16 bits and so for 32 interrupts it requires 64 memory locations and this is the this is the description of that.

(Refer Slide Time: 03:48)

Table 5. Interrupt Sources, Flags, and Vectors

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-Up External Reset Watchdog Timer+ Flash key violation PC out-of-range ⁽¹⁾	PORIFG RSTIFG WDTIFG KEYV ⁽²⁾	Reset	0FFFh	31, highest
NMI Oscillator fault Flash memory access violation	NMIIFG OFIFG ACCVIFG ⁽²⁾⁽³⁾	(non)-maskable (non)-maskable (non)-maskable	0FFCCh	30
Timer1_A3	TA1CCR0 CCFIFG ⁽⁴⁾	maskable	0FFFAh	29
Timer1_A3	TA1CCR2 TA1CCR1 CCIFG, TAIFG ⁽⁴⁾⁽⁵⁾	maskable	0FFF8h	28
Comparator_A+	CAIFG ⁽⁶⁾	maskable	0FFF6h	27
Watchdog Timer+	WDTIFG	maskable	0FFF4h	26
Timer0_A3	TA0CCR0 CCFIFG ⁽⁴⁾	maskable	0FFF2h	25
Timer0_A3	TA0CCR2 TA0CCR1 CCIFG, TAIFG (6)(4)	maskable	0FFF0h	24
USCI_A0/USCI_B0 receive USCI_B0 PC status	UCA0RXIFG, UCB0RXIFG ⁽⁷⁾⁽⁸⁾	maskable	0FFECh	23
USCI_A0/USCI_B0 transmit USCI_B0 PC receive/transmit	UCA0TXIFG, UCB0TXIFG ⁽⁷⁾⁽⁸⁾	maskable	0FFECh	22
ADC10 (MSP430G2x53 only)	ADC10IFG ⁽⁴⁾	maskable	0FFEAh	21
			0FFE8h	20
I/O Port P2 (up to eight flags)	P2IFG.0 to P2IFG.7 ⁽²⁾⁽⁴⁾	maskable	0FFE6h	19
I/O Port P1 (up to eight flags)	P1IFG.0 to P1IFG.7 ⁽²⁾⁽⁴⁾	maskable	0FFE4h	18
			0FFE2h	17
			0FFE0h	16
See ⁽⁷⁾			0FFDEh	15
See ⁽⁸⁾			0FFDEh to 0FFC0h	14 to 0, lowest

(1) A reset is generated if the CPU tries to fetch instructions from within the module register memory address range (0h to 01FFh) or from within unused address ranges.

As you can see at the top of the location is reset vector and you have the other interrupts associated with various peripherals of G2553. We will go in this details when we are using these

peripherals. So, I am not going through each one of them but as you can see from this table that it has interrupts for timers as well as comparators, watch dog timer, the universal serial communication interfaces and the EDCs and as well as ports.

(Refer Slide Time: 04:28)

Memory Organization (Information Memory)

Information memory is a 256B block of flash memory that is intended for storage of nonvolatile data.
 Memory address range is: 0x1000h to 0x10FFh.
 The information memory has 4 segments of 64 bytes each. Segment A contains factory calibration data for the DCO in the MSP430G2553. After reset, segment A is protected against programming and erasing.

0xFFFF	Interrupt and Reset Vector Table
0xFFC0	
0xFFBF	
	Flash Code Memory (lower boundary varies)
0xF000	Flash Information Memory
0xEFFF	
0x1100	
0x10FF	
0x1000	Bootstrap Loader (BSL)
0x0FFF	
0x0C00	RAM (upper boundary varies)
0x0BFF	
0x0280	
0x027F	
0x0200	peripheral registers (word access)
0x01FF	
0x0100	peripheral registers (byte access)
0x00FF	
0x000F	special function registers (byte access)
0x0000	

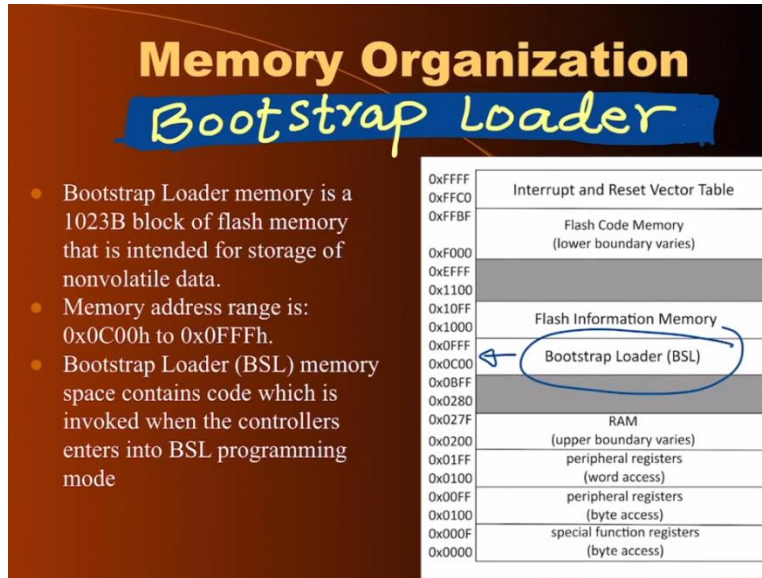
Now the information part information memory of the flash memory is in the case of G2553 you have 256 bytes here and it is starting from this address to this address. This is 256 bytes and the information these 256 bytes are splits in four segments of 64 bytes each as I mentioned earlier and these are used to store various information. You can store constants as well as the first segment is used to store factory calibration data for the digital controlled oscillator. The oscillator that uses the RC oscillators.

You want them to be accurate so each one of them is calibrated so that they provide your 16 megahertz or 8 Megahertz clock frequency and so that information is stored there and this segment called segment of the information memory is protected at reset. You cannot reset after reset you can overwrite this information because this is very critical information.

The next part of the memory is what we call as the bootstrap loader. Now I have mentioned in the past that MSP430 microcontroller can be programmed in several ways. One of them is to use this spy-bi-wire protocol and the other is through the J-Tag four J-Tag pins that we have on the micro controller like MSP430 and the third method is through the boot loader which in

application programming method of uploading a code and in MSP430 jargon this is called bootstrap loader.

(Refer Slide Time: 06:12)



And that bootstrap loader is located at this address as you see here. This is the location and in the case of MSP430 microcontroller you do not have to write that program. The each and each every MSP430 microcontroller specially the G series, they come with a preprogrammed bootstrap loader.

All you have to do is in your microcontroller system provide an ecosystem which is compatible with invoking the bootstrap loader mechanism and then you can upload any program from your desktop computer into the memory of the microcontroller and later in this section we will discuss about that but to mention here that about 1 kilobyte is flash memory is available and preprogrammed with a Bootloader provided by Texas Instruments.

(Refer Slide Time: 07:14)

Memory Organization (RAM)

- RAM is used for data/variables.
- RAM starts at 0200h. The end address of RAM depends on the amount of RAM present and varies by device.
- MSP430G2553 has 512 Bytes of RAM.

0xFFFF	Interrupt and Reset Vector Table
0xFFC0	
0xFFBF	Flash Code Memory (lower boundary varies)
0xF000	
0xEFFF	
0x1100	
0x10FF	Flash Information Memory
0x1000	
0x0FFF	Bootstrap Loader (BSL)
0x0C00	
0x0BFF	
0x0280	
0x027F	RAM (upper boundary varies)
0x0200	←
0x01FF	peripheral registers (word access)
0x0100	
0x00FF	peripheral registers (byte access)
0x0100	
0x000F	special function registers (byte access)
0x0000	

The RAM is used for storing variables as well as stacks and you have 512 bytes of S RAM available on the G255 series. Here is the address. This is the starting address 0200 and the upper limit of that address depends on exact chip number. In this case 512 bytes so from 0200 it will go up to 212, 512 locations.

(Refer Slide Time: 07:54)

Memory Organization (Peripheral Modules)

- Peripheral registers are used by CPU to access and configure peripherals.
- The address space from 0100 to 01FFh is reserved for 16-bit peripheral modules. These modules should be accessed with word instructions. If byte instructions are used, only even addresses are permissible, and the high byte of the result is always 0.
- You can find the details of each peripheral register and its address in the 'Peripheral File Map' in the datasheet.

0xFFFF	Interrupt and Reset Vector Table
0xFFC0	
0xFFBF	Flash Code Memory (lower boundary varies)
0xF000	
0xEFFF	
0x1100	
0x10FF	Flash Information Memory
0x1000	
0x0FFF	Bootstrap Loader (BSL)
0x0C00	
0x0BFF	
0x0280	
0x027F	RAM (upper boundary varies)
0x0200	
0x01FF	→ peripheral registers (word access)
0x0100	
0x00FF	→ peripheral registers (byte access)
0x0100	
0x000F	→ special function registers (byte access)
0x0000	

Apart from the flash, the information flash and the RAM, you also have memory used by the peripheral registers of the CPU and you can see them here. There are 3 types of resistors. Some of the resistors require 16 bit access so they are available here word access. Some of the

peripherals require 8 bit storage and so those resistors are accessible through this and then there are special function resistors which we will come to shortly.

(Refer Slide Time: 08:38)

Table 14. Peripherals With Word Access

MODULE	REGISTER DESCRIPTION	REGISTER NAME	OFFSET
ADC10 (MSP430G2x53 devices only)	ADC data transfer start address	ADC10SA	1BCh
	ADC memory	ADC10MEM	1B4h
	ADC control register 1	ADC10CTL1	1B2h
	ADC control register 0	ADC10CTL0	1B0h
Timer1_A3	Capture/compare register	TA1CCR2	0196h
	Capture/compare register	TA1CCR1	0194h
	Capture/compare register	TA1CCR0	0192h
	Timer_A register	TA1R	0190h
	Capture/compare control	TA1CCTL2	0186h
	Capture/compare control	TA1CCTL1	0184h
	Capture/compare control	TA1CCTL0	0182h
	Timer_A control	TA1CTL	0180h
	Timer_A interrupt vector	TA1IV	011Eh
	Timer0_A3	Capture/compare register	TA0CCR2
Capture/compare register		TA0CCR1	0174h
Capture/compare register		TA0CCR0	0172h
Timer_A register		TA0R	0170h
Capture/compare control		TA0CCTL2	0166h
Capture/compare control		TA0CCTL1	0164h
Capture/compare control		TA0CCTL0	0162h
Timer_A control		TA0CTL	0160h
Flash Memory	Timer_A interrupt vector	TA0IV	012Eh
	Flash control 3	FCTL3	012Ch
	Flash control 2	FCTL2	012Ah
	Flash control 1	FCTL1	0128h
Watchdog Timer+	Watchdog/timer control	WDCTL	0120h

This is the various 16 bit word storage peripheral resistors as you can see they deal with ADCs, they deal with timer 1 and timer 0 as well as flash memory. You can control access to the flash memory by writing into these resistors and you can also write to the control the watch dog timer at this location.

(Refer Slide Time: 09:05)

Memory Organization (Peripheral Modules)

- The address space from 010h to 0FFh is reserved for 8-bit peripheral modules. These modules should be accessed with byte instructions.
- Read access of byte modules using word instructions results in unpredictable data in the high byte. If word data is written to a byte module only the low byte is written into the peripheral register, ignoring the high byte.

0xFFFF	Interrupt and Reset Vector Table
0xFFC0	
0xFFBF	
0xF000	Flash Code Memory (lower boundary varies)
0xEFFF	
0x1100	Flash Information Memory
0x10FF	
0x1000	
0x0FFF	
0x0C00	Bootstrap Loader (BSL)
0x0BFF	
0x0280	RAM (upper boundary varies)
0x027F	
0x0200	
0x01FF	
0x0100	
0x00FF	
0x0100	peripheral registers (byte access)
0x000F	special function registers (byte access)
0x0000	

Then you have the peripheral modules for 8 Bit peripherals which is here and by reading and writing 8 bit data into this information you can control such peripherals.

(Refer Slide Time: 09:21)

Table 15. Peripherals With Byte Access				
MODULE	REGISTER DESCRIPTION	REGISTER NAME	OFFSET	
USCI_B0	USCI_B0 transmit buffer	UCB0TXBUF	06Fh	
	USCI_B0 receive buffer	UCB0RXBUF	06Eh	
	USCI_B0 status	UCB0STAT	06Dh	
	USCI_B0 I2C Interrupt enable	UCB0CIE	06Ch	
	USCI_B0 bit rate control 1	UCB0BR1	06Bh	
	USCI_B0 bit rate control 0	UCB0BR0	06Ah	
	USCI_B0 control 1	UCB0CTL1	069h	
	USCI_B0 control 0	UCB0CTL0	068h	
	USCI_B0 I2C slave address	UCB0SA	011Ah	
	USCI_B0 I2C own address	UCB0CA	0119h	
	USCI_A0	USCI_A0 transmit buffer	UCA0TXBUF	067h
	USCI_A0	USCI_A0 receive buffer	UCA0RXBUF	066h
USCI_A0 status		UCA0STAT	065h	
USCI_A0 modulation control		UCA0ACTL	064h	
USCI_A0 baud rate control 1		UCA0BR1	063h	
USCI_A0 baud rate control 0		UCA0BR0	062h	
USCI_A0 control 1		UCA0CTL1	061h	
USCI_A0 control 0		UCA0CTL0	060h	
USCI_A0 IrDA receive control		UCA0IRFCTL	05Fh	
USCI_A0 IrDA transmit control		UCA0IRTCTL	05Eh	
USCI_A0 auto baud rate control		UCA0ABCTL	05Dh	
ADC10 (MSP430G2x53 devices only)		ADC analog enable 0	ADC10AEN0	04Ah
		ADC analog enable 1	ADC10AEN1	04Bh
	ADC data transfer control register 1	ADC10DTC1	049h	
Comparator_A+	ADC data transfer control register 0	ADC10DTC0	048h	
	Comparator_A+ port disable	CAPD	05Bh	
Basic Clock System+	Comparator_A+ control 2	CACTL2	05Ah	
	Comparator_A+ control 1	CACTL1	059h	
	Basic clock system control 3	BCSCTL3	053h	
	Basic clock system control 2	BCSCTL2	058h	
Port P3 (28-pin PW and 32-pin RH8 only)	Basic clock system control 1	BCSCTL1	057h	
	DCO clock frequency control	DCOCTL	056h	
	Port P3 selection 2_pin	P3SEL2	043h	
	Port P3 resistor enable	P3REN	010h	
	Port P3 selection	P3SEL	018h	
	Port P3 direction	P3DIR	01A8h	
	Port P3 output	P3OUT	019h	
Port P2	Port P3 input	P3IN	018h	
	Port P2 selection 2	P2SEL2	042h	
	Port P2 resistor enable	P2REN	02Fh	
	Port P2 selection	P2SEL	02Eh	
	Port P2 interrupt enable	P2IE	02Dh	
	Port P2 interrupt edge select	P2IES	02Ch	
	Port P2 interrupt flag	P2IFG	02Bh	
	Port P2 direction	P2DIR	02Ah	
	Port P2 output	P2OUT	029h	

Here is a list as you can see there are much more than what you saw in the world accessible peripherals. This is the USCI that is Universal Serial Communication Interface. The second module you have some ADC devices, you have the comparator. The clock system can be controlled various low power modes by writing into this and the ports P3 and P2 can be accessed.

(Refer Slide Time: 09:48)

Table 15. Peripherals With Byte Access (continued)			
MODULE	REGISTER DESCRIPTION	REGISTER NAME	OFFSET
Port P1	Port P1 selection 2	P1SEL2	041h
	Port P1 resistor enable	P1REN	027h
	Port P1 selection	P1SEL	026h
	Port P1 interrupt enable	P1IE	025h
	Port P1 interrupt edge select	P1IES	024h
	Port P1 interrupt flag	P1IFG	023h
	Port P1 direction	P1DIR	022h
	Port P1 output	P1OUT	021h
Special Function	Port P1 input	P1IN	020h
	SFR interrupt flag 2	IFG2	003h
	SFR interrupt flag 1	IFG1	002h
	SFR interrupt enable 2	IE2	001h
	SFR interrupt enable 1	IE1	000h

And similarly Port 1 and other special function resistors related to the interrupt control are accessible through this byte access special function resistors. Well I meant byte accessed peripheral resistors.

(Refer Slide Time: 10:02)

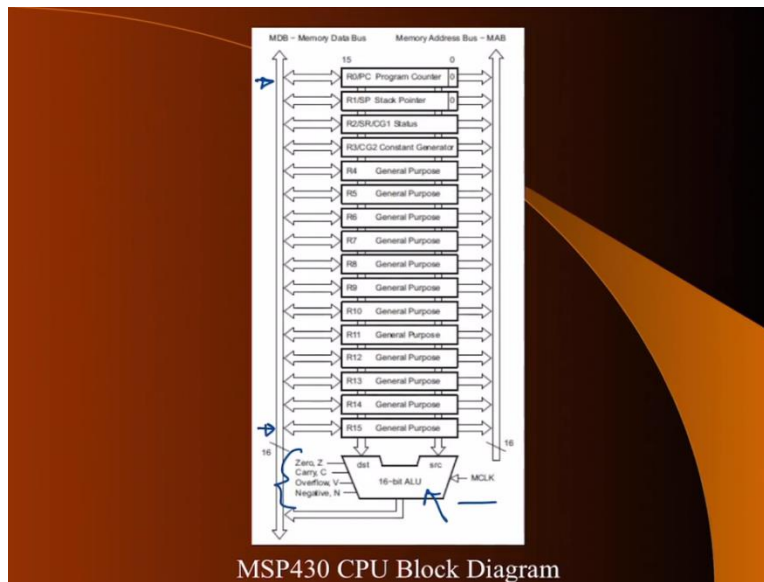
Memory Organization (Special Function Registers)

- Some peripheral functions are configured in the SFRs. The SFRs are located in the lower 16 bytes of the address space, and are organized by byte. SFRs must be accessed using byte instructions only.
- MSP430G2553 has interrupt enables and interrupt flag registers as SFRs.

0xFFFF	Interrupt and Reset Vector Table
0xFFC0	
0xFFBF	Flash Code Memory (lower boundary varies)
0xF000	
0xEFFF	
0x1100	
0x10FF	Flash Information Memory
0x1000	
0x0FFF	
0x0C00	Bootstrap Loader (BSL)
0x0BFF	
0x0280	
0x027F	RAM (upper boundary varies)
0x0200	
0x01FF	peripheral registers (word access)
0x0100	
0x00FF	peripheral registers (byte access)
0x0100	
0x000F	special function registers (byte access)
0x0000	

Then we have special function resistors, these deals with various enabling of interrupts and allowing interrupts to function and flag resistors related to them. The storage is 16 locations that is 16 bytes. Now we have reached a stage where we are ready to talk about the CPU in MSP430. This CPU of MSP430 is RISC architecture and as recommended by the risk specifications which were brought forward in early 80's. RISC wanted all the CPU to have large number of resistors so that local variables could be stored within the CPU rather than having to go in the external memory such as the RAM for fetching them which slows down the execution

(Refer Slide Time: 11:09)



And in keeping with that recommendation MSP430 has 16 general purpose resistors. They have labeled from R0 here to R15. Each of these resistors can store 16 bit of data. These resistor feed 16 bit ALU as you see here. The ALU can perform various mathematical and logical operations like add, subtract and AND/OR operations and the result of these operations can be observed through these flags but these flags are accessible through one of these general purpose resistors that we will see. So, we have in all 16 resistors. Let us see the function of these resistors.

(Refer Slide Time: 11:52)

CPU Registers

16 Registers: The generous set of 16 registers is characteristic of a RISC CPU. These Registers provide reduced instruction execution time. Register to register operation takes only clock cycle.

4 Special Purpose

- Program Counter(PC) ←
- Stack Pointer(SP) ←
- Status Register(SR) ←
- Constant Generator ←

12 General Purpose ←

Program Counter	PC/R0
Stack Pointer	SP/R1
Status Register	SR/CG1/R2
Constant Generator	CG2/R3
General-Purpose Register	R4
General-Purpose Register	R5
General-Purpose Register	R6
General-Purpose Register	R7
General-Purpose Register	R8
General-Purpose Register	R9
General-Purpose Register	R10
General-Purpose Register	R11
General-Purpose Register	R12
General-Purpose Register	R13
General-Purpose Register	R14
General-Purpose Register	R15

CPU Registers

There are 4 special function resistors which means that out of this 16, 4 are taken off because they have dedicated function. One of the resistors is the program counter which means that the content of this resistor will tell you which memory location in flash the microcontroller CPU is fetching the current instruction from. The second resistor is for stack pointer which tells you that where is the stack located in the RAM.

It is your responsibility to ensure that the stack pointer is pointing in the right memory of the RAM and the third is the status resistor which indicates the various flags that we just saw whether the result of a particular CPU operation of AND (12.36) whether the result is 0 or whether there is a overflow. All those flags will be available in this status resistor and there is interesting concept called the constant generator.

In many programs you require certain constants and this constant generator offers you that flexibility without having to store them in your program and then apart from that you are left with twelve general purpose resistors which the CPU utilizes. Now since we are not going to program MSP430 in our course in assembly language program, we do not really worry about how this general purpose resistor are going to be used.

It is depended on the C compiler or whichever compiler that we use to optimally use these resistors but to know that they exist is great consolation that our program is going to be compiled efficiently.

(Refer Slide Time: 13:33)

CPU Registers

Program Counter

- This contains the address of the next instruction to be executed. Instructions are composed of 1-3 words, which must be aligned to even addresses, so the LSB of the PC is hard-wired to 0.

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Program Counter Bits 15 to 1

0

0

The slide features a dark blue background with a light blue curved graphic element. The title 'CPU Registers' is in a large, bold, light blue font. Below it, the sub-section 'Program Counter' is underlined. A bullet point explains its function and the hard-wired LSB. At the bottom, a bit diagram shows bits 15 to 1 in a white box, with bit 0 circled in red and containing a '0'.

So, let us see which special registers we have. We have a program counter and since the program counter deals with reading the next instruction. The instructions are all 16 bits wide whereas our memories are arranged in terms of bytes and therefore the least significant bit here is set to 0 because we want to store all our instructions at even byte addresses even addresses.

So, our first instruction if it is at 0 the next will be 2 and then next will be at 4 and this can be ensured by ensuring the least significant bit is set to 0 and therefore out of potential 64 kilobyte memory space we can actually store how many instructions? We can store half of them 32768 total instructions you can store.

Of course, the actual number of instructions that you store will depend on the available memory. The flash memory that you have but this kind of limits the maximum number of instructions that you can have. Instructions themselves can be one word which means 2 bytes or two words or three words that is depends on the exact instruction.

(Refer Slide Time: 14:55)

CPU Registers

Stack Pointer

- The stack pointer is used by the CPU to store the return addresses of subroutine calls and interrupts.
- The SP is initialized into RAM by the user, and is aligned to even addresses.
- SP holds the address of the most recently added word in the stack and is automatically adjusted as the stack goes up and down.

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Stack Pointer Bits 15 to 1

0

The slide features a dark background with a light-colored curved shape on the right. The title 'CPU Registers' is in a large, bold, yellow font. Below it, 'Stack Pointer' is underlined in yellow. Three bullet points describe the stack pointer's function. At the bottom, a bit diagram shows bits 15 to 1 of the stack pointer, with bit 0 circled in blue and containing the value 0.

Then the second register we have is a stack pointer. A stack pointer is a special register which points to the stack and stack is that memory area where you are going to store your return address and you call a subroutine or where the CPU stores the return address in case an (instruction) interrupt subroutine gets called.

So this is the location, this is the RAM location in which you store your return address. Again just like the program counter, the insignificant bit is set to 0 so that it is able to write 16 bit

numbers which refer to the address of your program the return address and therefore this bit is 0 the rest of the bits are available to you and of course it your responsibility to initialize the stack pointer appropriately.

(Refer Slide Time: 15:44)

CPU Registers

Status Register

- This contains a set of flags (single bits), whose functions fall into three categories.
 - The most commonly used flags are C, Z, N, and V, which give information about the result of the last arithmetic or logical operation. Decisions that affect the flow of control in the program can be made by testing these bits.
 - Setting the GIE bit enables maskable interrupts.
 - The final group of bits is CPUOFF, OSCOFF, SCG0, and SCG1, which control the mode of operation of the MCU.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reserved							V	SCG1	SCG0	OSCOFF	CPU OFF	GIE	N	Z	C	
rw-0							rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

The third resistor is a status resistor and the status resistor consists of various flags and we will see the details of it. It also has access to the interrupt mask so that you can enable disable interrupts and it has some special flags which allow you to turn the CPU off or turn the oscillator off and so on and we will see these operations subsequently in our exercises.

(Refer Slide Time: 16:10)

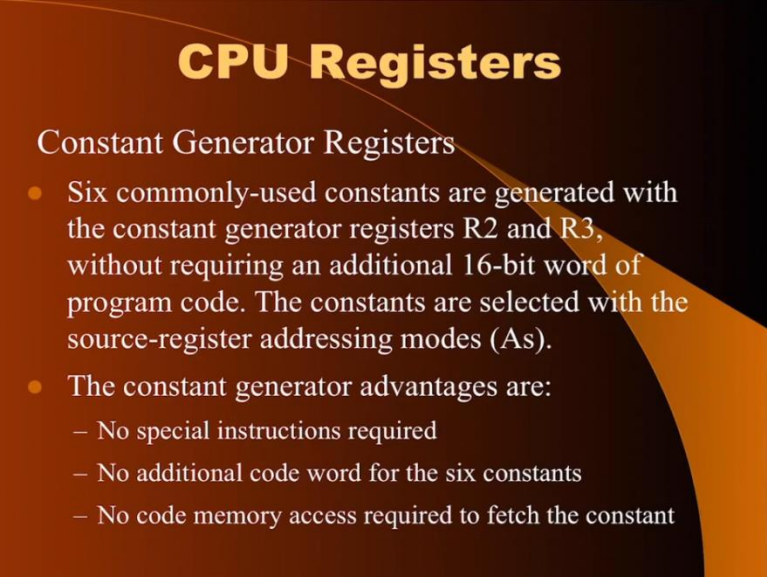
Status Register

Bit	Description
V	<p>Overflow bit. This bit is set when the result of an arithmetic operation overflows the signed-variable range.</p> <p>ADD (.B), ADDC (.B)</p> <p style="margin-left: 40px;">Set when:</p> <p style="margin-left: 80px;">Positive + Positive = Negative</p> <p style="margin-left: 80px;">Negative + Negative = Positive</p> <p style="margin-left: 80px;">Otherwise reset</p> <p>SUB (.B), SUBC (.B), CMP (.B)</p> <p style="margin-left: 40px;">Set when:</p> <p style="margin-left: 80px;">Positive - Negative = Negative</p> <p style="margin-left: 80px;">Negative - Positive = Positive</p> <p style="margin-left: 80px;">Otherwise reset</p>
SCG1	System clock generator 1. When set, turns off the SMCLK.
SCG0	System clock generator 0. When set, turns off the DCO dc generator, if DCOCLK is not used for MCLK or SMCLK.
OSCOFF	Oscillator Off. When set, turns off the LFXT1 crystal oscillator, when LFXT1CLK is not use for MCLK or SMCLK.
CPUOFF	CPU off. When set, turns off the CPU.
GIE	General interrupt enable. When set, enables maskable interrupts. When reset, all maskable interrupts are disabled.
N	Negative bit. Set when the result of a byte or word operation is negative and cleared when the result is not negative.
Z	Zero bit. Set when the result of a byte or word operation is 0 and cleared when the result is not 0.
C	Carry bit. Set when the result of a byte or word operation produced a carry and cleared when no carry occurred.

We have overflow flag which is set to 1 when the result is an overflow as a result of mathematical operations when you are dealing with sine bit numbers. We have these two system clock generator bits where you can turn on and turn off the main clock or the other clocks. We have the oscillator clocks and so on and here is the general purpose interruptible and then these are the 3 flags apart from the overflow flag which is the negative flag which means that when the result the most significant bit of your number that you are looking at if it is 1 this is negative flag you have the 0 flag.

When you do any mathematical operation or logical operations it results in 0 in all the bits and a carry flag when there is carry from the most significant bit into the outside that. So these are the flags that are available.

(Refer Slide Time: 17:09)



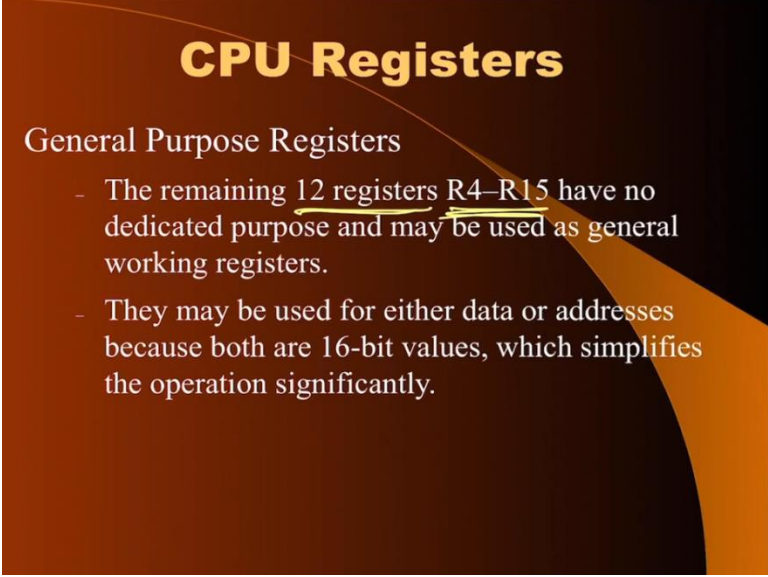
CPU Registers

Constant Generator Registers

- Six commonly-used constants are generated with the constant generator registers R2 and R3, without requiring an additional 16-bit word of program code. The constants are selected with the source-register addressing modes (As).
- The constant generator advantages are:
 - No special instructions required
 - No additional code word for the six constants
 - No code memory access required to fetch the constant

The constant generator registers register is able to provide 6 commonly used commonly used values using R2 and R3 and no special instruction is required to access those constant values and we look at a program so that we illustrate this concept.

(Refer Slide Time: 17:31)



CPU Registers

General Purpose Registers

- The remaining 12 registers R4–R15 have no dedicated purpose and may be used as general working registers.
- They may be used for either data or addresses because both are 16-bit values, which simplifies the operation significantly.

The rest of the general purpose resistors can store 16 bit values and you have 12 resistors to do that as you see here from R4 to R15 and there is no dedicated function associated with these resistors. These will depend on if you are going to program in assembly language then it is your responsibility to use them judiciously. If you are going to use a high level language compiler then it is compiler responsibility to use these resistors as it may deem fit.

Now, since we are not going to program MSP430 in assembly language programming but we still want to have a feel of the kind of instructions that are available as I mentioned MSP430 is RISC architecture which means the number of instructions expected is less compared to corresponding CISC architecture.

It may have varieties of those basic instructions in the form of addressing modes so we will just go through those names and we will see how much time some these instruction take and what is the size of these instructions as I mentioned the instructions could be 1 word or 2 words or 3 words and then eventually we will see how we are going to be experimenting with MSP430 microcontroller in our course by looking at the design of the MSP430 lunchbox.

(Refer Slide Time: 18:53)

Instruction Format

There are three formats of instructions in the MSP430:

- ✎ **Double operand:** Arithmetic and logical operations with two operands such as ADD R4,R5. Both operands must be specified in the instruction. This contrasts with accumulator-based architectures, where an accumulator or working register is used automatically as the destination and one operand.
- **Single operand:** A mixture of instructions for control or to manipulate a single operand, which is effectively the *source* for the addressing modes.
- **Jumps:** The jump to the destination rather than its absolute address, in other words the offset that must be added to the program counter.








INSTRUCTION FORMAT	EXAMPLE	OPERATION
Dual operands, source-destination	ADD R4,R5	$R4 + R5 \rightarrow R5$
Single operands, destination only	CALL R8 ↵	$PC \rightarrow (TOS), R8 \rightarrow PC$
Relative jump, un/conditional ↵	JNE	Jump-on-equal bit = 0

So, the instructions are they can have double operands here such that you want to add the value of as an example R4 and R5. Here the sum of R4 and R5 which are general purpose resistor is stored in resistor R5 here. This is the destination.

You can have a single operand instruction where apart from the instruction the opcode you have a operand here you are saying call subroutine which is located at an address whose address is located at a flash address whose the address which is stored in a register R8 and you could have another instruction such as a jump instruction where it is unconditional jump with a relative which means compare to current location you can jump forward or you can jump backward.

(Refer Slide Time: 19:46)

Addressing Modes

- Addressing modes are the ways in which operands can be specified.
- MSP430 has seven addressing modes. These are:
 1. Register Mode 
 2. Indexed Mode 
 3. Symbolic Mode 
 4. Absolute Mode 
 5. Indirect Register Mode 
 6. Indirect Autoincrement Mode 
 7. Immediate Mode 

These are the 7 addressing modes which on the basic operation you can the addressing mode specifies how do we get the operands and so we have a resistor mode in which 2 resistor or 1 resistor is used to specify the operand. You have indexed mode which means one the resistor is used as an address to go into a memory location to fetch the operand. You have symbolic mode which is referring to the labels that you would use in a program jump here or call that subroutine at that location.

You have absolute mode where you are specifying the entire value. You have indirect resistor mode which means you are going to use resistor in a indirect mode. You can do an auto increment on those resistor values and you could use an immediate value such as you want to initialize the resistor with some number. These are 7 addressing modes that operate on the basic instructions and create variety for you.

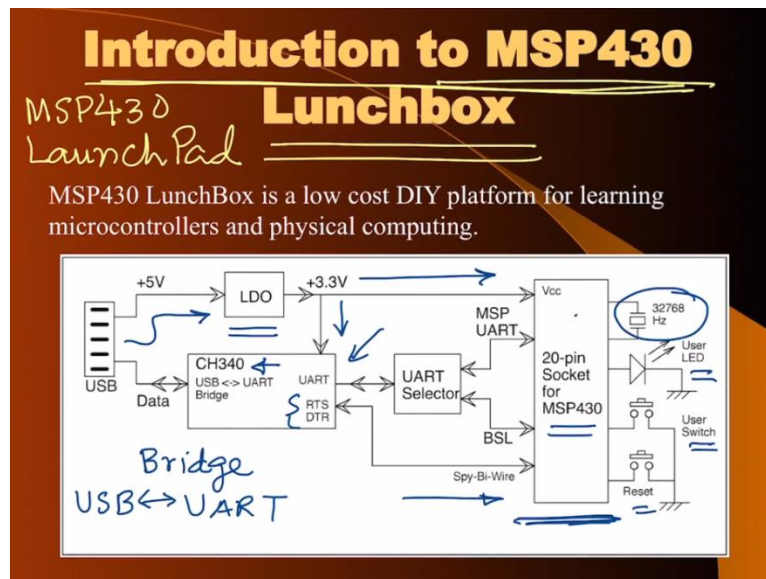
(Refer Slide Time: 20:48)

Source/Destination Operand Addressing Modes

As/Ad	Addressing Mode	Syntax	Description
00/0	Register mode	Rn	Register contents are operand
01/1	Indexed mode	X(Rn)	(Rn + X) points to the operand. X is stored in the next word.
01/1	Symbolic mode	ADDR	(PC + X) points to the operand. X is stored in the next word. Indexed mode X(PC) is used.
01/1	Absolute mode	&ADDR	The word following the instruction contains the absolute address. X is stored in the next word. Indexed mode X(SR) is used.
10/-	Indirect register mode	@Rn	Rn is used as a pointer to the operand.
11/-	Indirect autoincrement	@Rn+	Rn is used as a pointer to the operand. Rn is incremented afterwards by 1 for .B instructions and by 2 for .W instructions.
11/-	Immediate mode	#N	The word following the instruction contains the immediate constant N. Indirect autoincrement mode @PC+ is used.

And this is the addressing the source and destination bits that specify each of these modes as you can see. We are not going to again go into the details of this because as an embedded system designer we would worry about other issues.

(Refer Slide Time: 21:08)



Now we are we have reached a point where we are now ready to jump into experimenting with MSP430 microcontroller and for that as you see as I have mentioned you mentioned here several times we have designed microcontroller evaluation kit called as MSP430 lunchbox. Now let me also briefly mention that this is not the only evaluation kit that you have. That Texas instruments

makes several, several wonderful evaluation kit and the most popular is called MSP430 launch pad that is that is their official evaluation kit.

This kit is priced very, very competitively and if you get your hands on it you will realize that on this kit there are actually two microcontrollers. One of the microcontroller is the one that you are going to use which we call as the target microcontroller and there is a control microcontroller which controls this target microcontroller and helps you download program into it or to emulate it or to debug it and so it is a extremely competitive and extremely versatile piece of hardware.

Unfortunately it is slightly more expensive than what is possible to make with microcontroller by itself and this where we come into picture and we have designed this lunch box. As you see here is our target microcontroller which is going to be MSP430 G2553 in the in the dip format. Now I have been mentioning this several times in the past that there are several ways of programming MSP430 microcontroller.

One is through these Spy-bi-wire methods which incidentally is the method that official Texas instruments MSP430 launch pad uses. The other method is by accessing the J tag interface pins four of them. TDI TDO TMS and TCK. The third method is using what is called as the in application programming what Texas instruments calls as a bootstrap loader and the advantage of that is that you do not need any special mechanism to transfer program from your development host which is the desktop or laptop computer into the memory of the microcontroller.

All you need is knowledge of which signals to activate and a mechanism to connect the desktop computer and microcontroller and today as you know one of the most common methods of connecting anything to the desktop computer or laptop computer is through the USB. Unfortunately the G2553 microcontroller does not have USB interface.

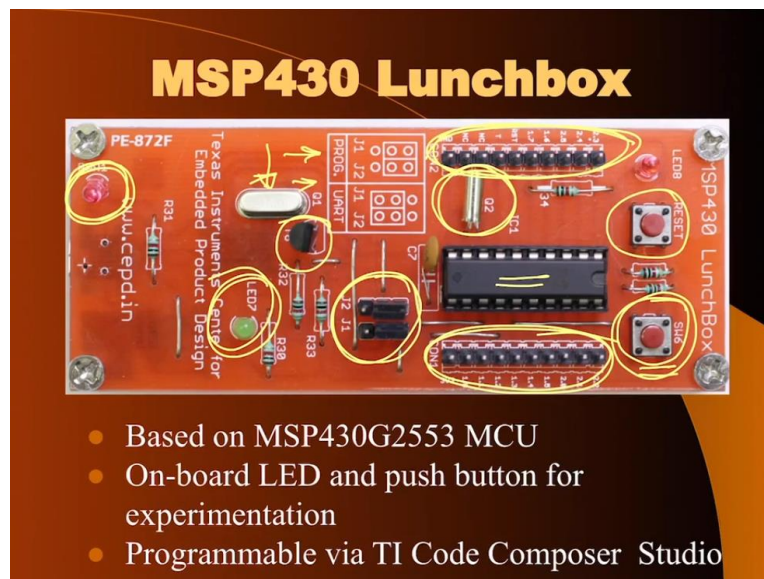
It has UART interface so you have to have a mechanism by which the USB signals from the desktop or laptop computer could be translated and converted into UART and such a device which converts from one protocol into another has a name and it is called a bridge. A bridge is simply a mechanism to go from one side of the river to other side likewise I would say we need a device which is USB to UART Bridge.

Now there are several mechanisms to achieve this bridge functionality and we have used IC called CH340. This is commonly available bridge integrated circuit. On one side its talks to the connects to the USB and on the other side it has the UART functions and apart from that it has to control signals.

One is called request to send and Data DTR signals and these are used to connect to certain pins of the MSP430 G2553 series microcontroller. In this Spy-bi-wire mode only for control and then because you are MSP430 microcontroller operates at 3.3 volts or up to 3.6 volts and not 5 volts, we have taken the 5 volts available on the USB passed it through this low drop out voltage regulator to provide 3.3 volts supply voltage not only to the micro controller the target microcontroller but also to the bridge.

Apart from that you have 32.768 kilohertz crystal which is used to generate the bitrate what is called as board rate for UART communication. You have a user LED, you have one user switch, you have a reset switch other than that rest of the available pins are accessible through headed pins as we will see in the next slide.

(Refer Slide Time: 26:15)



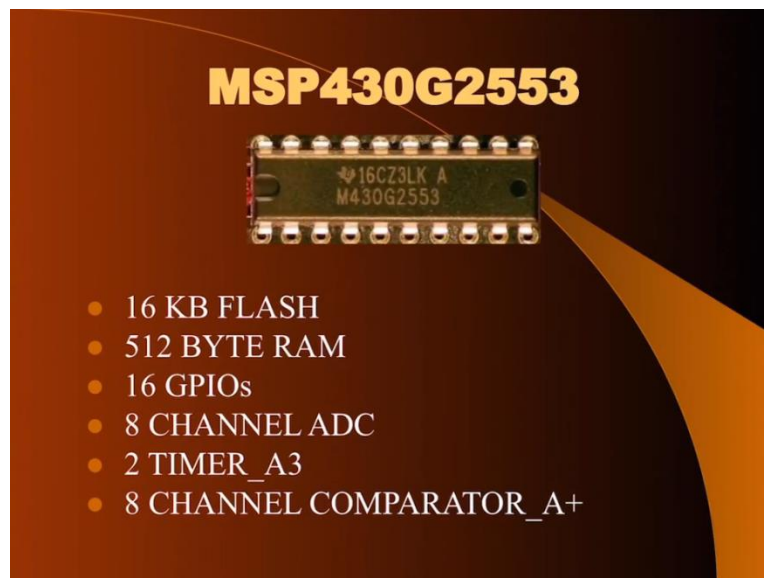
Here is the photograph of the MSP430 lunch box and let me show you this is the microcontroller MSP430 G2553. Here is the reset switch. Here is the user switch. We have the power on LED here and a power on LED is very important component to have whether you are designing a evaluation kit or your own instrument because that is the only way to convey to the user that the

power to the system is on if it was not there and if the system is not working the user would be confused is it working because the power has not been applied or whether it is not working because the system is faulty.

So, power on indicator is a good way for that. You also have this crystal which is required this crystal is required by this CH340 USB to UART bridge. You also have 32 kilohertz crystal as you saw in the previous slide. This is used by MSP430 for UART communication. The available pins that MSP430 offers you are available on these header pins and you have this is the user LED here. This is the User switch that you can that is connected to certain pins of the resident microcontroller and here are some jumpers you have to engage in certain way as mentioned here.

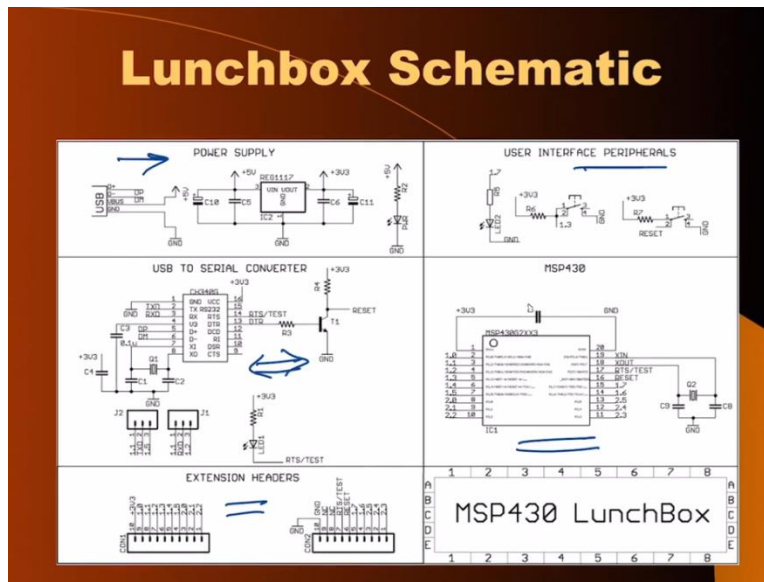
If you want to put the MSP430 kit into programming mode which means you want to download code then this jumpers have to be engaged. These are shorting jumper with which you can connect two neighboring pins. You have to engage them in certain way and if you want to use the same USB interface for serial communication in your program through your program during runtime then you change the setting of these jumpers and this mentioned here and here. And this is a transistor which is required for some level inversion.

(Refer Slide Time: 28:15)



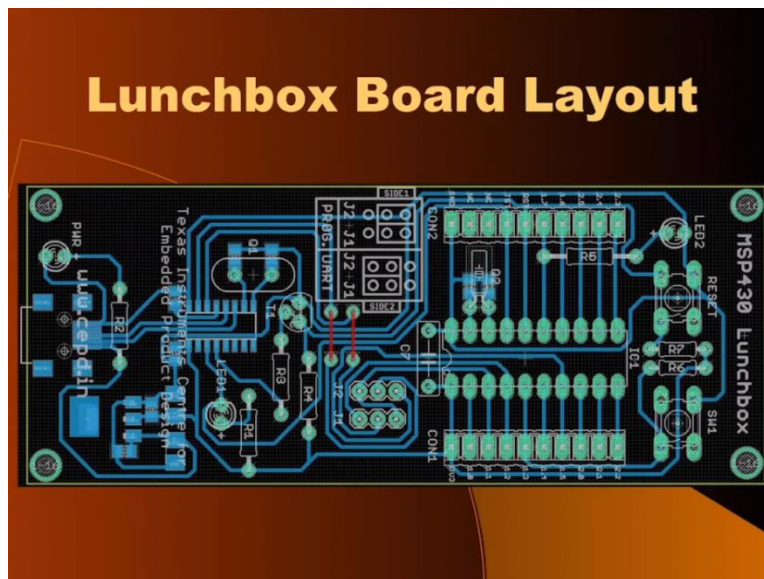
This is the microcontroller that we are using. It has all these features which we have gone through before.

(Refer Slide Time: 28:21)



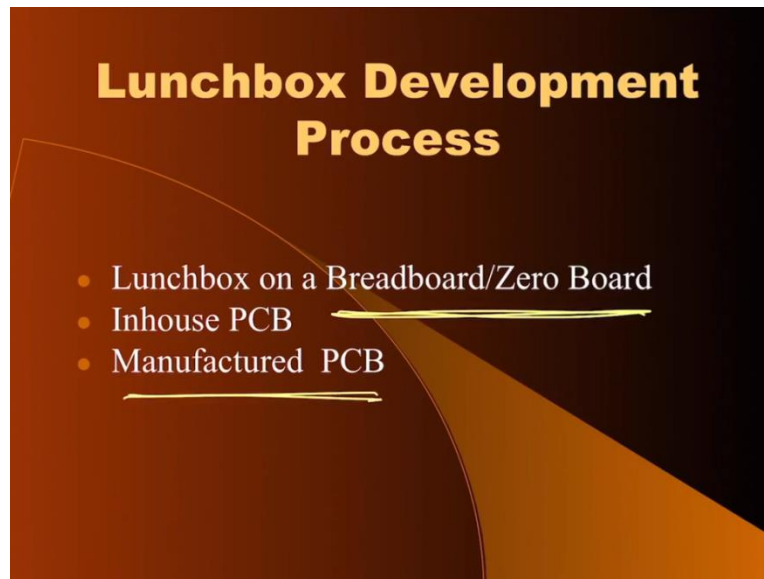
This is the schematic I encourage you to go through this schematic it has been annotated nicely. Here is the power supply. Here is the user peripherals which is switch and LED. Here is a USB to UART Bridge. Here is your MSP430 and here you have extension headers.

(Refer Slide Time: 28:40)



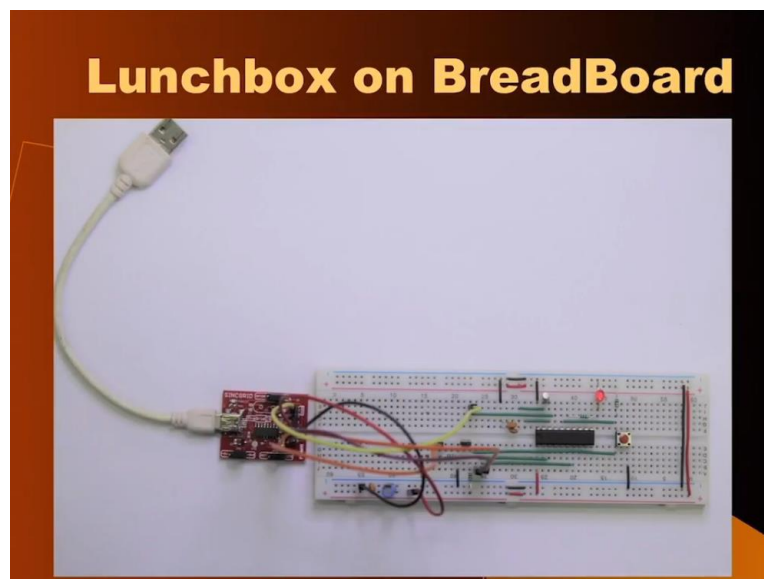
This is the layout and as I mentioned repeatedly that this layout is available on my blog website which we have shared on the GIT website. You can download this and you can print your own printed circuit board and solder all the things yourself and you can make this evaluation kit yourself.

(Refer Slide Time: 29:01)



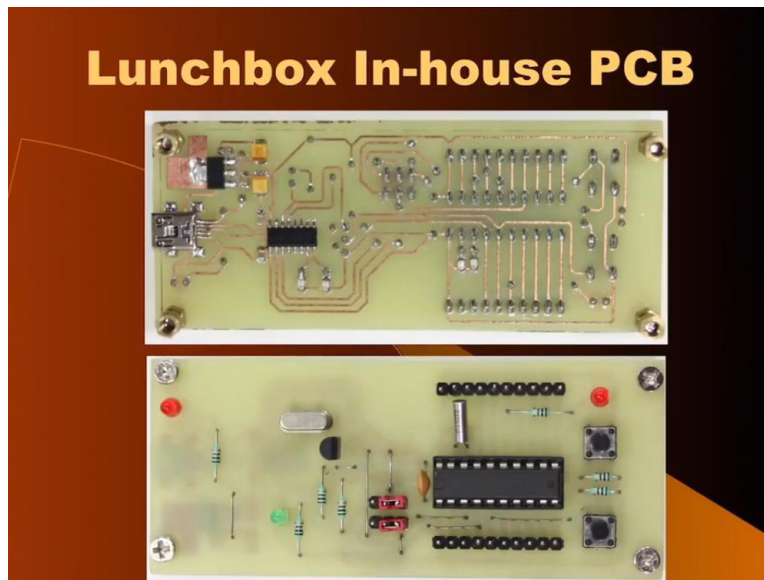
If you do not have it you can even temporarily put together all the components on a bread board or a 0 board. You can have a in-house you can make you own PCB or you can get a manufactured PCB from outside.

(Refer Slide Time: 29:16)



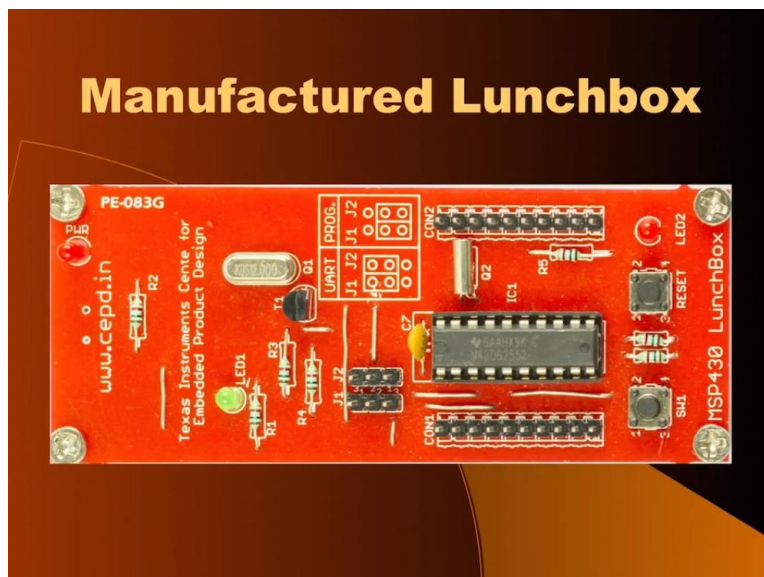
And so this is the evaluation the lunch box fabricated on a bread board.

(Refer Slide Time: 29:22)



This is in-house that in my lab we used earlier.

(Refer Slide Time: 29:27)



And this is the manufactured lunch box. I hope that you have access to one of these three lunch boxes to be able to continue having extensive hands on sessions in future. I will see you very soon. Thank very much.